```mql5
//+------------------------------------------------------------------+
//|                                 MorningEvening StarDoji RSI.mq5 |
//|                                 Copyright 2024, MetaQuotes Ltd. |
//|                                             https://www.mql5.com |
//+------------------------------------------------------------------+
#property copyright "Copyright 2024, MetaQuotes Ltd."
#property link      "https://www.mql5.com"
#property version   "1.00"

#include <Trade\Trade.mqh>
#include <Trade\SymbolInfo.mqh>

#define SIGNAL_BUY    1             // Buy signal
#define SIGNAL_NOT    0             // no trading signal
#define SIGNAL_SELL  -1             // Sell signal

#define CLOSE_LONG    2             // signal to close Long
#define CLOSE_SHORT  -2             // signal to close Short

//--- Input parameters
input int InpAverBodyPeriod=12;     // period for calculating average candlestick size
input int InpPeriodRSI     =37;     // RSI period
input ENUM_APPLIED_PRICE  InpPrice=PRICE_CLOSE;    // price type

//--- trade parameters
input uint InpDuration=10;          // position holding time in bars
input uint InpSL      =200;         // Stop Loss in points
input uint InpTP      =200;         // Take Profit in points
input uint InpSlippage=10;          // slippage in points
//--- money management parameters
input double InpLot=0.1;            // lot
//--- Expert ID
input long InpMagicNumber=130300;   // Magic Number

//--- global variables
int    ExtAvgBodyPeriod;            // average candlestick calculation period
int    ExtSignalOpen     =0;        // Buy/Sell signal
int    ExtSignalClose    =0;        // signal to close a position
string ExtPatternInfo    ="";       // current pattern information
string ExtDirection      ="";       // position opening direction
bool   ExtPatternDetected=false;    // pattern detected
bool   ExtConfirmed      =false;    // pattern confirmed
bool   ExtCloseByTime    =true;     // requires closing by time
bool   ExtCheckPassed    =true;     // status checking error
//---  indicator handle
int    ExtIndicatorHandle=INVALID_HANDLE;

//--- service objects
CTrade      ExtTrade;
CSymbolInfo ExtSymbolInfo;
//+------------------------------------------------------------------+
//| Expert initialization function                                   |
//+------------------------------------------------------------------+
int OnInit()
  {
   Print("InpSL=", InpSL);
   Print("InpTP=", InpTP);
//--- set parameters for trading operations
   ExtTrade.SetDeviationInPoints(InpSlippage);    // slippage
   ExtTrade.SetExpertMagicNumber(InpMagicNumber); // Expert Advisor ID
   ExtTrade.LogLevel(LOG_LEVEL_ERRORS);           // logging level

   ExtAvgBodyPeriod=InpAverBodyPeriod;
//--- indicator initialization
   ExtIndicatorHandle=iRSI(_Symbol, _Period, InpPeriodRSI, InpPrice);
   if(ExtIndicatorHandle==INVALID_HANDLE)
     {
      Print("Error creating CCI indicator");
      return(INIT_FAILED);
     }
//--- OK
   return(INIT_SUCCEEDED);
  }
//+------------------------------------------------------------------+
//| Expert deinitialization function                                 |
//+------------------------------------------------------------------+
void OnDeinit(const int reason)
  {
//--- release indicator handle
   IndicatorRelease(ExtIndicatorHandle);
  }
//+------------------------------------------------------------------+
//| Expert tick function                                             |
//+------------------------------------------------------------------+
```

```
void OnTick()
  {
//--- save the next bar start time; all checks at bar opening only
   static datetime next_bar_open=0;

//--- Phase 1 - check the emergence of a new bar and update the status
   if(TimeCurrent()>=next_bar_open)
     {
      //--- get the current state of environment on the new bar
      // namely, set the values of global variables:
      // ExtPatternDetected - pattern detection
      // ExtConfirmed - pattern confirmation
      // ExtSignalOpen - signal to open
      // ExtSignalClose - signal to close
      // ExtPatternInfo - current pattern information
      if(CheckState())
        {
         //--- set the new bar opening time
         next_bar_open=TimeCurrent();
         next_bar_open-=next_bar_open%PeriodSeconds(_Period);
         next_bar_open+=PeriodSeconds(_Period);

         //--- report the emergence of a new bar only once within a bar
         if(ExtPatternDetected && ExtConfirmed)
            Print(ExtPatternInfo);
        }
      else
        {
         //--- error getting the status, retry on the next tick
         return;
        }
     }

//--- Phase 2 - if there is a signal and no position in this direction
   if(ExtSignalOpen && !PositionExist(ExtSignalOpen))
     {
      Print("\r\nSignal to open position ", ExtDirection);
      PositionOpen();
      if(PositionExist(ExtSignalOpen))
         ExtSignalOpen=SIGNAL_NOT;
     }

//--- Phase 3 - close if there is a signal to close
   if(ExtSignalClose && PositionExist(ExtSignalClose))
     {
      Print("\r\nSignal to close position ", ExtDirection);
      CloseBySignal(ExtSignalClose);
      if(!PositionExist(ExtSignalClose))
         ExtSignalClose=SIGNAL_NOT;
     }

//--- Phase 4 - close upon expiration
   if(ExtCloseByTime && PositionExpiredByTimeExist())
     {
      CloseByTime();
      ExtCloseByTime=PositionExpiredByTimeExist();
     }
  }
//+------------------------------------------------------------------+
//|  Get the current environment and check for a pattern             |
//+------------------------------------------------------------------+
bool CheckState()
  {
//--- check if there is a pattern
   if(!CheckPattern())
     {
      Print("Error, failed to check pattern");
      return(false);
     }

//--- check for confirmation
   if(!CheckConfirmation())
     {
      Print("Error, failed to check pattern confirmation");
      return(false);
     }
//--- if there is no confirmation, cancel the signal
   if(!ExtConfirmed)
      ExtSignalOpen=SIGNAL_NOT;

//--- check if there is a signal to close a position
   if(!CheckCloseSignal())
     {
      Print("Error, failed to check the closing signal");
```

```
      return(false);
     }

//--- if positions are to be closed after certain holding time in bars
   if(InpDuration)
      ExtCloseByTime=true; // set flag to close upon expiration

//--- all checks done
   return(true);
  }
//+------------------------------------------------------------------+
//| Open a position in the direction of the signal                   |
//+------------------------------------------------------------------+
bool PositionOpen()
  {
   ExtSymbolInfo.Refresh();
   ExtSymbolInfo.RefreshRates();

   double price=0;
//--- Stop Loss and Take Profit are not set by default
   double stoploss=0.0;
   double takeprofit=0.0;

   int    digits=ExtSymbolInfo.Digits();
   double point=ExtSymbolInfo.Point();
   double spread=ExtSymbolInfo.Ask()-ExtSymbolInfo.Bid();

//--- uptrend
   if(ExtSignalOpen==SIGNAL_BUY)
     {
      price=NormalizeDouble(ExtSymbolInfo.Ask(), digits);
      //--- if Stop Loss is set
      if(InpSL>0)
        {
         if(spread>=InpSL*point)
           {
            PrintFormat("StopLoss (%d points) <= current spread = %.0f points. Spread value will be used", InpSL, spread/
point);
            stoploss = NormalizeDouble(price-spread, digits);
           }
         else
            stoploss = NormalizeDouble(price-InpSL*point, digits);
        }
      //--- if Take Profit is set
      if(InpTP>0)
        {
         if(spread>=InpTP*point)
           {
            PrintFormat("TakeProfit (%d points) < current spread = %.0f points. Spread value will be used", InpTP, spread/
point);
            takeprofit = NormalizeDouble(price+spread, digits);
           }
         else
            takeprofit = NormalizeDouble(price+InpTP*point, digits);
        }

      if(!ExtTrade.Buy(InpLot, Symbol(), price, stoploss, takeprofit))
        {
         PrintFormat("Failed %s buy %G at %G (sl=%G tp=%G) failed. Ask=%G error=%d",
                     Symbol(), InpLot, price, stoploss, takeprofit, ExtSymbolInfo.Ask(), GetLastError());
         return(false);
        }
     }

//--- downtrend
   if(ExtSignalOpen==SIGNAL_SELL)
     {
      price=NormalizeDouble(ExtSymbolInfo.Bid(), digits);
      //--- if Stop Loss is set
      if(InpSL>0)
        {
         if(spread>=InpSL*point)
           {
            PrintFormat("StopLoss (%d points) <= current spread = %.0f points. Spread value will be used", InpSL, spread/
point);
            stoploss = NormalizeDouble(price+spread, digits);
           }
         else
            stoploss = NormalizeDouble(price+InpSL*point, digits);
        }
      //--- if Take Profit is set
      if(InpTP>0)
        {
         if(spread>=InpTP*point)
```

```
            {
             PrintFormat("TakeProfit (%d points) < current spread = %.0f points. Spread value will be used", InpTP, spread/
point);
             takeprofit = NormalizeDouble(price-spread, digits);
            }
          else
             takeprofit = NormalizeDouble(price-InpTP*point, digits);
        }

      if(!ExtTrade.Sell(InpLot, Symbol(), price,  stoploss, takeprofit))
        {
         PrintFormat("Failed %s sell at %G (sl=%G tp=%G) failed. Bid=%G error=%d",
                     Symbol(), price, stoploss, takeprofit, ExtSymbolInfo.Bid(), GetLastError());
         ExtTrade.PrintResult();
         Print("   ");
         return(false);
        }
     }

   return(true);
  }
//+------------------------------------------------------------------+
//|  Close a position based on the specified signal                  |
//+------------------------------------------------------------------+
void CloseBySignal(int type_close)
  {
//--- if there is no signal to close, return successful completion
   if(type_close==SIGNAL_NOT)
      return;
//--- if there are no positions opened by our EA
   if(PositionExist(ExtSignalClose)==0)
      return;

//--- closing direction
   long type;
   switch(type_close)
     {
      case CLOSE_SHORT:
         type=POSITION_TYPE_SELL;
         break;
      case CLOSE_LONG:
         type=POSITION_TYPE_BUY;
         break;
      default:
         Print("Error! Signal to close not detected");
         return;
     }

//--- check all positions and close ours based on the signal
   int positions=PositionsTotal();
   for(int i=positions-1; i>=0; i--)
     {
      ulong ticket=PositionGetTicket(i);
      if(ticket!=0)
        {
         //--- get the name of the symbol and the position id (magic)
         string symbol=PositionGetString(POSITION_SYMBOL);
         long   magic =PositionGetInteger(POSITION_MAGIC);
         //--- if they correspond to our values
         if(symbol==Symbol() && magic==InpMagicNumber)
           {
            if(PositionGetInteger(POSITION_TYPE)==type)
              {
               ExtTrade.PositionClose(ticket, InpSlippage);
               ExtTrade.PrintResult();
               Print("   ");
              }
           }
        }
     }
  }
//+------------------------------------------------------------------+
//|  Close positions upon holding time expiration in bars            |
//+------------------------------------------------------------------+
void CloseByTime()
  {
//--- if there are no positions opened by our EA
   if(PositionExist(ExtSignalOpen)==0)
      return;

//--- check all positions and close ours based on the holding time in bars
   int positions=PositionsTotal();
   for(int i=positions-1; i>=0; i--)
     {
```

```
      ulong ticket=PositionGetTicket(i);
      if(ticket!=0)
        {
         //--- get the name of the symbol and the position id (magic)
         string symbol=PositionGetString(POSITION_SYMBOL);
         long   magic =PositionGetInteger(POSITION_MAGIC);
         //--- if they correspond to our values
         if(symbol==Symbol() && magic==InpMagicNumber)
           {
            //--- position opening time
            datetime open_time=(datetime)PositionGetInteger(POSITION_TIME);
            //--- check position holding time in bars
            if(BarsHold(open_time)>=(int)InpDuration)
              {
               Print("\r\nTime to close position #", ticket);
               ExtTrade.PositionClose(ticket, InpSlippage);
               ExtTrade.PrintResult();
               Print("    ");
              }
           }
        }
     }
  }
//+------------------------------------------------------------------+
//| Returns true if there are open positions                         |
//+------------------------------------------------------------------+
bool PositionExist(int signal_direction)
  {
   bool check_type=(signal_direction!=SIGNAL_NOT);

//--- what positions to search
   ENUM_POSITION_TYPE search_type=WRONG_VALUE;
   if(check_type)
      switch(signal_direction)
        {
         case SIGNAL_BUY:
            search_type=POSITION_TYPE_BUY;
            break;
         case SIGNAL_SELL:
            search_type=POSITION_TYPE_SELL;
            break;
         case CLOSE_LONG:
            search_type=POSITION_TYPE_BUY;
            break;
         case CLOSE_SHORT:
            search_type=POSITION_TYPE_SELL;
            break;
         default:
            //--- entry direction is not specified; nothing to search
            return(false);
        }

//--- go through the list of all positions
   int positions=PositionsTotal();
   for(int i=0; i<positions; i++)
     {
      if(PositionGetTicket(i)!=0)
        {
         //--- if the position type does not match, move on to the next one
         ENUM_POSITION_TYPE type=(ENUM_POSITION_TYPE)PositionGetInteger(POSITION_TYPE);
         if(check_type && (type!=search_type))
            continue;
         //--- get the name of the symbol and the expert id (magic number)
         string symbol =PositionGetString(POSITION_SYMBOL);
         long   magic  =PositionGetInteger(POSITION_MAGIC);
         //--- if they correspond to our values
         if(symbol==Symbol() && magic==InpMagicNumber)
           {
            //--- yes, this is the right position, stop the search
            return(true);
           }
        }
     }

//--- open position not found
   return(false);
  }
//+------------------------------------------------------------------+
//| Returns true if there are open positions with expired time       |
//+------------------------------------------------------------------+
bool PositionExpiredByTimeExist()
  {
//--- go through the list of all positions
   int positions=PositionsTotal();
```

```
   for(int i=0; i<positions; i++)
     {
      if(PositionGetTicket(i)!=0)
        {
         //--- get the name of the symbol and the expert id (magic number)
         string symbol =PositionGetString(POSITION_SYMBOL);
         long   magic  =PositionGetInteger(POSITION_MAGIC);
         //--- if they correspond to our values
         if(symbol==Symbol() && magic==InpMagicNumber)
           {
            //--- position opening time
            datetime open_time=(datetime)PositionGetInteger(POSITION_TIME);
            //--- check position holding time in bars
            int check=BarsHold(open_time);
            //--- id the value is -1, the check completed with an error
            if(check==-1 || (BarsHold(open_time)>=(int)InpDuration))
               return(true);
           }
        }
     }

//--- open position not found
   return(false);
  }
//+------------------------------------------------------------------+
//| Checks position closing time in bars                             |
//+------------------------------------------------------------------+
int BarsHold(datetime open_time)
  {
//--- first run a basic simple check
   if(TimeCurrent()-open_time<PeriodSeconds(_Period))
     {
      //--- opening time is inside the current bar
      return(0);
     }
//---
   MqlRates bars[];
   if(CopyRates(_Symbol, _Period, open_time, TimeCurrent(), bars)==-1)
     {
      Print("Error. CopyRates() failed, error = ", GetLastError());
      return(-1);
     }
//--- check position holding time in bars
   return(ArraySize(bars));
  }
//+------------------------------------------------------------------+
//| Returns the open price of the specified bar                      |
//+------------------------------------------------------------------+
double Open(int index)
  {
   double val=iOpen(_Symbol, _Period, index);
//--- if the current check state was successful and an error was received
   if(ExtCheckPassed && val==0)
      ExtCheckPassed=false;   // switch the status to failed

   return(val);
  }
//+------------------------------------------------------------------+
//| Returns the close price of the specified bar                     |
//+------------------------------------------------------------------+
double Close(int index)
  {
   double val=iClose(_Symbol, _Period, index);
//--- if the current check state was successful and an error was received
   if(ExtCheckPassed && val==0)
      ExtCheckPassed=false;   // switch the status to failed

   return(val);
  }
//+------------------------------------------------------------------+
//| Returns the low price of the specified bar                       |
//+------------------------------------------------------------------+
double Low(int index)
  {
   double val=iLow(_Symbol, _Period, index);
//--- if the current check state was successful and an error was received
   if(ExtCheckPassed && val==0)
      ExtCheckPassed=false;   // switch the status to failed

   return(val);
  }
//+------------------------------------------------------------------+
//| Returns the high price of the specified bar                      |
//+------------------------------------------------------------------+
```

```
double High(int index)
  {
   double val=iHigh(_Symbol, _Period, index);
//--- if the current check state was successful and an error was received
   if(ExtCheckPassed && val==0)
      ExtCheckPassed=false;   // switch the status to failed

   return(val);
  }
//+------------------------------------------------------------------+
//| Returns the middle body price for the specified bar              |
//+------------------------------------------------------------------+
double MidPoint(int index)
  {
   return(High(index)+Low(index))/2.;
  }
//+------------------------------------------------------------------+
//| Returns the middle price of the range for the specified bar      |
//+------------------------------------------------------------------+
double MidOpenClose(int index)
  {
   return((Open(index)+Close(index))/2.);
  }
//+------------------------------------------------------------------+
//| Returns the average candlestick body size for the specified bar  |
//+------------------------------------------------------------------+
double AvgBody(int index)
  {
   double sum=0;
   for(int i=index; i<index+ExtAvgBodyPeriod; i++)
     {
      sum+=MathAbs(Open(i)-Close(i));
     }
   return(sum/ExtAvgBodyPeriod);
  }
//+------------------------------------------------------------------+
//| Returns true in case of successful pattern check                 |
//+------------------------------------------------------------------+
bool CheckPattern()
  {
   ExtPatternDetected=false;
//--- check if there is a pattern
   ExtSignalOpen=SIGNAL_NOT;
   ExtPatternInfo="\r\nPattern not detected";
   ExtDirection="";

//--- check Evening Doji
   if((Close(3)-Open(3)>AvgBody(1))              && // bullish candlestick, its body is larger than average
      (MathAbs(Close(2)-Open(2))<AvgBody(1)*0.1)  &&
// second candlestick body is doji (less than one tenth of the average candle body)
      (Close(2)>Close(3))                        && // second candlestick close is higher than first candlestick close
      (Open(2)>Open(3))                          && // second candlestick open is higher than first candlestick open
      (Open(1)<Close(2))                         && // down price gap on the last candlestick
      (Close(1)<Close(2)))                          // last candlestick close lower than second candlestick close
     {
      ExtPatternDetected=true;
      ExtSignalOpen=SIGNAL_SELL;
      ExtPatternInfo="\r\nEvening Doji detected";
      ExtDirection="Sell";
      return(true);
     }

//--- check Evening Star
   if((Close(3)-Open(3)>AvgBody(1))              && // bullish candlestick, its body is larger than average
      (MathAbs(Close(2)-Open(2))<AvgBody(1)*0.5) &&
// second candlestick body is short (less than a half of the average candle body)
      (Close(2)>Close(3))                        && // second candlestick close is higher than first candlestick close
      (Open(2)>Open(3))                          && // second candlestick open is higher than first candlestick open
      (Close(1)<MidOpenClose(3)))
// last candlestick close is lower than the middle of the first (bullish) one
     {
      ExtPatternDetected=true;
      ExtSignalOpen=SIGNAL_SELL;
      ExtPatternInfo="\r\nEvening Star detected";
      ExtDirection="Sell";
      return(true);
     }

//--- check Morning Doji
   if((Open(3)-Close(3)>AvgBody(1))              && // bearish candlestick, its body is larger than average
      (MathAbs(Close(2)-Open(2))<AvgBody(1)*0.1) &&
// second candlestick body is doji (less than one tenth of the average candle body)
      (Close(2)<Close(3))                        && // second candlestick close is lower than first candlestick close
      (Open(2)<Open(3))                          && // second candlestick open is lower than first candlestick open
```

```
      (Open(1)>Close(2))                        && // upward price gap on the last candlestick
      (Close(1)>Close(2)))                         // last candlestick close higher than second candlestick close
     {
      ExtPatternDetected=true;
      ExtSignalOpen=SIGNAL_BUY;
      ExtPatternInfo="\r\nMorning Doji detected";
      ExtDirection="Buy";
      return(true);
     }

//--- check Morning Star
   if((Open(3)-Close(3)>AvgBody(1))             && // bearish candlestick, its body is larger than average
      (MathAbs(Close(2)-Open(2))<AvgBody(1)*0.5) &&
// second candlestick body is short (less than a half of the average candle body)
      (Close(2)<Close(3))                       && // second candlestick close is lower than first candlestick close
      (Open(2)<Open(3))                         && // second candlestick open is lower than first candlestick open
      (Close(1)>MidOpenClose(3)))
// last candlestick close is lower than the middle of the first (bearish) one
     {
      ExtPatternDetected=true;
      ExtSignalOpen=SIGNAL_BUY;
      ExtPatternInfo="\r\nMorning Star detected";
      ExtDirection="Buy";
      return(true);
     }

//--- result of checking
   return(ExtCheckPassed);
  }
//+------------------------------------------------------------------+
//| Returns true in case of successful confirmation check            |
//+------------------------------------------------------------------+
bool CheckConfirmation()
  {
   ExtConfirmed=false;
//--- if there is no pattern, do not search for confirmation
   if(!ExtPatternDetected)
      return(true);

//--- get the value of the stochastic indicator to confirm the signal
   double signal=RSI(1);
   if(signal==EMPTY_VALUE)
     {
      //--- failed to get indicator value, check failed
      return(false);
     }

//--- check the Buy signal
   if(ExtSignalOpen==SIGNAL_BUY && (signal<40))
     {
      ExtConfirmed=true;
      ExtPatternInfo+="\r\n   Confirmed: RSI<40";
     }

//--- check the Sell signal
   if(ExtSignalOpen==SIGNAL_SELL && (signal>60))
     {
      ExtConfirmed=true;
      ExtPatternInfo+="\r\n   Confirmed: RSI>60";
     }

//--- successful completion of the check
   return(true);
  }
//+------------------------------------------------------------------+
//| Check if there is a signal to close                              |
//+------------------------------------------------------------------+
bool CheckCloseSignal()
  {
   ExtSignalClose=false;
//--- if there is a signal to enter the market, do not check the signal to close
   if(ExtSignalOpen!=SIGNAL_NOT)
      return(true);

//--- check if there is a signal to close a long position
   if(((RSI(1)<70) && (RSI(2)>70)) || ((RSI(1)<30) && (RSI(2)>30)))
     {
      //--- there is a signal to close a long position
      ExtSignalClose=CLOSE_LONG;
      ExtDirection="Long";
     }

//--- check if there is a signal to close a short position
   if(((RSI(1)>30) && (RSI(2)<30)) || ((RSI(1)>70) && (RSI(2)<70)))
```

```
     {
      //--- there is a signal to close a short position
      ExtSignalClose=CLOSE_SHORT;
      ExtDirection="Short";
     }

//--- successful completion of the check
   return(true);
  }
//+------------------------------------------------------------------+
//| RSI indicator value at the specified bar                         |
//+------------------------------------------------------------------+
double RSI(int index)
  {
   double indicator_values[];
   if(CopyBuffer(ExtIndicatorHandle, 0, index, 1, indicator_values)<0)
     {
      //--- if the copying fails, report the error code
      PrintFormat("Failed to copy data from the RSI indicator, error code %d", GetLastError());
      return(EMPTY_VALUE);
     }
   return(indicator_values[0]);
  }
//+------------------------------------------------------------------+
```